# HOW TO DEFINE A LANGUAGE

## Examples

Throughout this text reference is made to the following standards:

1. The COMAL 80 standard
2. ANSI standard 'USA Standard FORTRAN X3.9-1966'

## The style of the standard

A fundamental difference between these standards is that COMAL is defined using a top-down method, whereas FORTRAN is defined bottom-upwards. An important decision that needs to be made at an early stage is whether to define COMAL 86 using top-down or bottom-up techniques.

A major problem with top-down analysis is that it does not encourage the author to produce a *complete* and rigorous standard, whereas in bottom-up analysis completeness is easily checked for the following reason: at the first level, independent fundamental elements of the language are defined. Terms introduced in subsequent levels must have been defined previously. Thus the completeness of the standard is ensured once the basic level has been established. In bottom-up analysis, this basic level is defined *first* whereas in top-down it is defined last (*or not at all*).

Top-down analysis might be preferred because it is easier to use for reference purposes, since it is more natural to think in a top-down fashion. Hence it may be desirable to include a brief top-down summary of the language as an appendix. Note that this would note be part of the standard, but mereley an aid to interpreting it.

Taking these factors into consideration, the authors feel that COMAL 86 should be defined using bottom-up analysis. This is in direct contrast to the existing COMAL 80 standard.

## The structure of the standard

The standard will consist of the kernel, defined in the bottom-up manner outlined above, together with definitions of various options associated with packages. Provision must be made in the kernel for the incorporation of additional packages without producing contradictions.

This can be achieved by specifying that the standard consists of the kernel and *standard* packages. A form is in the standard if is in any of the standard packages or the kernel. No package may redefine the meaning of any form in the kernel, and no two packages may provide different interpretations for the same form. Since the packages may not all be present in any particular implementation, two or more packages may provide the same interpretation of any form.

Each of the packages will be defined in the same manner as the kernel and may rely on items defined in the kernel. However the packages themselves must be independent. Within individual packages different levels of implementation may be incorporated; consider for example a graphics package. Various levels of implementation may be:

Level 1. Monochrome character graphics only
Level 2. Monochrome line drawing.
Level 3. Colour line drawing.
Level 4. Colour line drawing and area filling.

Levels must be upwards compatible, so that a level 4 graphics implementation will run any program produced on a level 1,2,3 or 4 package.

One problem immediately arises with this levelled approach. Consider another graphics example:

The following features are available:

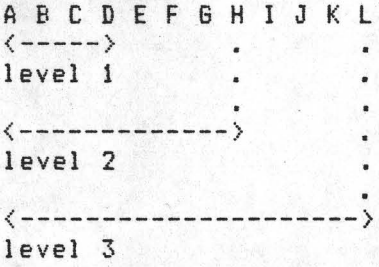
A. Line drawing
B. Hardware sprites
C. Polygon filling routines

Consider four hypothetical machines:

P has A only
Q has A and B
R has A and C
S has A, B and C

The graphics level 1 can be defined has having A only. However defining the next level is not easy since features B and C are of comparable sophistication level. The next level could be defined as having all three features. But then what do machines Q and R say about their implementations ?

This problem tends to suggest that all of the features that are defined in any package should be completely ordered. For instance the features could be




An implementation will be described itself as the highest level for which it has all the features. I.e. one with A-H, J and K will be 'a level 2 implementation with level 3 partially implemented' so that a user of it will know that any program written on a level 1 or level 2 system will run on his system, and one written on a

level 3 system might work. The ordering should be done according to degree of sophistication. In cases where features are of comparable sophistication the decision of ordering is arbitrary but must still be made. Another possibility is to say that a package is implemented 'up to feature H'. This would be equivalent to introducing a new level for every feature but may not be feasible.

The scope of the definition

As well as defining the actual language, the standard should also include various aspects of using the language, for instance if a program that creates files is to be portable, then there must be at least some agreement on filenames. This may well necessitate each machine setting up prefixes to specify devices, etc, in commands separate to the totally standard language commands. e.g the OPEN command may say OPEN FILE 1,"data" which would be portable through all machines, but to make this work with the hardware there may well need to be a DEVICE command, e.g DEVICE ":A." or DEVICE "£27" where commands such as DEVICE would have to be left very loosely defined in terms of the arguments they can take so that all machines can accommodate them.

This last example shows the need for a class of non-portable commands (unless a way round this can be discovered). The ways these can be included in programs to make them as portable and as easy to use as possible will require careful consideration.

There also will need to be a class of undefined features. This will include such things as maximum program size, precision of arithmetic, etc. Many of these system dependent constants will need to be accessed; for instance an advanced maths package will require the accuracy of calculations as a system constant.

This completes the discussion of the methods suggested for defining the standard. The following section is a brief attempt at starting a definition of the language kernel:

# THE COMAL 86 KERNEL

Note: Throughout the remainder of this document, the word 'COMAL' implies the phrase 'COMAL 86'

## 1. Purpose and Scope

### 1.1 Purpose

This standard defines the form of and the interpretation of programs written in the COMAL language for the purpose of promoting a high degree of portability of such programs for use on a variety of computers and microcomputers. A computer or microcomputer shall conform to this standard provided it accepts, and interprets as specified, at least those features described herein.

Any statement of requirement by the standard can be replaced by one saying that the standard does not specify an interpretation unless

the requirement is met. Further, any statement of prohibition could be replaced by one saying that the standard provides no interpretation when the prohibition is violated. When no interpretation of a form is provided then an implementation may provide any interpretation of the form insofar as the interpretation of the COMAL forms and relationships described are not affected.

### 1.2 Scope

1.2.1 This standard establishes:
(1) The form of a program written in the COMAL language
(2) The form of commands given directly to the COMAL system and of data and program lines to be input to the system.
(3) Rules for interpreting the meaning of such a program
(4) Minimum accuracies and ranges for numerical quantities
(5) The form of the output resulting from the use of such a program, provided that the rules of interpretation establish an interpretation.

1.2.2 This standard does not prescribe:
(1) The mechanism by which a computer or microcomputer executes COMAL programs, provided the rules for interpretation are complied with.
(2) The size or complexity of programs that will exceed the capacity of any specific computer or microcomputer

## 2. Basic Terminology

This section introduces some basic terminology and some concepts. They are defined more rigorously in later sections.

This section depends so greatly on the decided allowable forms for COMAL 86 that no attempt can sensibly be made to write it before the form of COMAL 86 has been discussed by the standards committee.

## 3. Program Form

### 3.1 Character Set

**Note:** all characters in the input to a COMAL system may be subject to preprocessing by the system which will affect such things as case of letters, etc. See section n.n.n for details.
The processed character set consists of the follwing characters:
A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z,0,1,2,3,4,5,6,7,8,9 and

| Character | Name of Character |
|---|---|
| | Space |
| ! | Pling |
| ? | Query |
| = | Equals |
| + | Plus |
| - | Minus |
| * | Asterisk |
| / | Slash |
| " | Double Quotation Mark |
| ' | Single Quotation Mark |

| | |
|---|---|
| $ | Dollar |
| £ | Hash |
| % | Cent |
| ^ | Up Arrow |
| ( | Left Parenthesis |
| ) | Right Parenthesis |
| < | Less Than |
| > | Greater Than |
| . | Decimal Point |
| , | Comma |
| : | Colon |
| ; | Semi-colon |
| _ | Underscore |
| & | Ampersand |
| ~ | Tilde |
| @ | At sign |

**3.1.1 Digits**

**3.1.1.1 Decimal Digits.** A decimal digit is on of the ten characters 0,1,2,3,4,5,6,7,8,9. Unless otherwise specified, where a string of digits requires interpretation as a number it shall be interpreted in the decimal number system.

**3.1.1.2 Hexadecimal Digits.** A hexadecimal digit is one of the sixteen characters 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

**3.1.2 Letters**

A letter is one of the 26 characters A,B,C,D,E,F,G,H,I,J,K,L,M, N,O,P,Q,R,S,T,U,V,W,X,Y,Z.

**3.1.3 Alphanumeric Characters.**

An alphanumeric character is a letter, a decimal digit or an underscore character.

**3.1.4 Special Characters**

A special character is one of the twenty six characters mentioned above.

**3.1.4.1** *Space character*

With the exception of the uses specified in sections n.n.n m.m.m and x.x.x the space character may be inserted anywhere in a program line to improve readability and layout.

## 3.2 Ordering of Characters

An ordering of characters is assumed within a program. Thus any meangingful collection of characters that constitutes names, lines or statements exists as a totally ordered set. This order is imposed by the order in which characters are passed into the COMAL system.

## 3.3 Symbolic Names

A symbolic name is a sequence of at least one alphanumeric character, the first of which must be a letter.

*Note that the COMAL standard may require to define symbolic names with type suffices, e.g £ or $ etc.*

## 4. Data Types

Four data types are defined. These are real, integer, byte, and string types. Each type has a different significance in terms of the datum it supplies to the expression in which it appears. It also determines the type of datum required to be assigned to the variable.

### 4.1 Data Type Association.

*Since the rules for this in COMAL 80 caused considerable controversy, this initial draft will not attempt to define them*

### 4.2 Data Type Properties.

The mathematical and the representation properties for each of the data types are defined in this section. The value zero is considered to be neither positive or negative.

**4.2.1 Real Type.** A real datum is an approximation to the value of a real number. It should be stored in floating point form. It may assume positive, negative or zero values.

**4.2.2 Integer Type.** An integer datum is always an exact representation of an integer value. It may assume positive, negative or zeo values. It may assume only integral values.

**4.2.3 Byte Type.** A byte datum is always an exact representation of an integer value. It is intended to have a range at least as small as the integer type an should be stored in the smallest amount of memory possible. It may assume positive and zero values only.

**4.2.4 String Type.** A string datum is a ordered sequence of characters which are not necessarily in the COMAL character set.

### 4.3 Statements

A statement is any string of characters which can validly form an entire program line.

### 4.4 Lines

A line is a string of up to 128 characters. All characters must be part of the COMAL character set except as specified in n.n.n and m.m.m

---

Reasons of time mean that the rest of the draft kernel will not be included in this preliminary discussion document. There should be ample material already in the document to give an idea of the technique suggested. Readers are once again suggested to look at the ANSI FORTRAN 66 standard. Any enquiries regarding this discussion document should be sent to:

David Christensen,
32, Brookfield Gardens,
Sarisbury Green,
Southampton. SO3 6DT
ENGLAND

Tel: 04895-84068